Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Discrete Algorithms 3 (2005) 101–112
JOURNAL OF DISCRETE ALGORITHMS
www.elsevier.com/locate/jda

Virtual path layouts optimizing total hop count on ATM tree networks

David Peleg [a,*,1], Uri Pincas [b,2]

[a] *Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel*

[b] *Cohn Institute, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel*

Available online 24 June 2004

Abstract

This paper presents some algorithmic results concerning virtual path layouts for the one-to-many communication problem in ATM tree networks. The ATM network model is based on covering the network with a layout of virtual paths, under some constraints on the allowed load, namely, the number of paths that can share an edge. The quality measure used is the hop count, namely, the number of edges traversed between two vertices that need to communicate. Whereas most former results concerned the maximum hop count of the virtual path layout, our interest here is in measuring its total hop count, or alternatively its average hop count. The paper presents a dynamic programming algorithm for planning ATM network layouts with minimal total hop count for one-to-many requirements under load constraints over the class of tree networks.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Virtual paths; Hop count; ATM networks; Network design; Dynamic programming

1. Introduction

This paper concerns the problem of designing efficient virtual path layouts on optical or ATM networks (see, e.g., [11,12,15,16]). In the ATM model, the routing and message forwarding tasks on a given network are simplified by predefining a collection of *virtual*

* Corresponding author.
E-mail addresses: david.peleg@weizmann.ac.il (D. Peleg), pincasur@post.tau.ac.il (U. Pincas).

[1] Supported in part by grants from the Israel Science Foundation and the Israel Ministry of Science and Art.

[2] Work was done while the author was with the Department of Mathematics and Computer Science, Bar Ilan University, Ramat Gan 52900, Israel.

1570-8667/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2004.05.003

paths (*VP's*), which are simple paths in the network, and performing end-to-end communication over routes composed of a sequence of such VP's (i.e., using the VP's as basic segments within complete routes). An elegant formulation of the problem can be obtained by representing the VP's formed on the given communication network G as a virtual graph H over the same set of vertices. Specifically, a VP connecting v and u in G is represented by an edge connecting v and u in the virtual graph H. The pair (H, P), where P is the collection of VP's corresponding to the edges of H, is referred to as the *virtual path layout* (*VPL*) for the physical graph G. Each route in G can be viewed as a simple path in the virtual graph H.

Various formulations of the VPL problem attempt to design a system of virtual paths which optimizes some parameters of the system while meeting some given communication demands between pairs of nodes and satisfying certain prespecified constraints. Usually, the problem is studied against one canonical pattern of communication demands, known as *all-to-all* communication, which requires communication between every pair of vertices in the network.

Research on virtual path layouts has concentrated on optimizing two central parameters of conflicting nature. The first is the *load* of a physical edge, defined as the number of VP's that share it. The upper bound on the allowed load of an edge is termed the *capacity* of the edge. The *maximum* (*respectively, average*) *load* of a VPL (H, P), denoted $\mathcal{L}_{\max}(H, P)$ (respectively, $\mathcal{L}_{\text{avg}}(H, P)$), is defined as the maximum (respectively, average) load of the edges in the network. These parameters determine the size of the VP routing tables, and reflect the traffic load on the links.

The second parameter of interest concerns the *hop count*, namely, the number of VP's occurring on the routes of the VPL, or equivalently, the number of edges in the corresponding path on the virtual graph H. Expressed in terms of H, the *maximum* hop count, denoted $\mathcal{H}_{\max}(H)$, can be viewed as the *diameter* of H, and the *average* hop count, denoted $\mathcal{H}_{\text{avg}}(H)$, can be defined as the average distance over all vertex pairs in H. Equivalently, one may consider the *total* number of hop counts over all vertex pairs in H, termed the *total hop count* of H and denoted $\mathcal{H}_{\text{tot}}(H)$. (Clearly, $\mathcal{H}_{\text{tot}}(H) = \mathcal{H}_{\text{avg}}(H) \cdot n(n-1)/2$ for every graph H.) These parameters measure the (worst-case or average) efficiency of setting up the route, and also the overall (worst-case or average) delay incurred by the route in a model where the processing along VP's is negligible compared to the processing at the VP endpoints. See [16] for a discussion of these parameters and their significance.

A number of studies have tackled the VPL problem (cf. [1,5,10,14]). The problem of minimizing $\mathcal{H}_{\max}(H)$, the diameter of a virtual graph H, subject to a specified upper bound c on the maximum load of the VPL, $\mathcal{L}_{\max}(H, P) \leqslant c$, has been considered in the undirected case in [6,8–10,13,16]. Conversely, the problem of minimizing the maximum load $\mathcal{L}_{\max}(H, P)$ over all VPL's (H, P) with maximum hop count bounded by h, $\mathcal{H}_{\max}(H) \leqslant h$, is studied in [2,7]. Minimizing also the average load $\mathcal{L}_{\text{avg}}(H, P)$ is considered in [9].

As links based on optical fibers are directed, and may have a different load in the two directions, it may be useful to consider a directed model as in [3,4], rather than an undirected one. The problem of minimizing the diameter $\mathcal{H}_{\max}(H)$ of a virtual directed graph (digraph) H over bounded maximum load VPL's is considered in [3], giving lower and upper bounds on the virtual diameter $\mathcal{H}_{\max}(H)$ of a directed VPL (henceforth, DVPL) with a prespecified capacity c bounding the maximum load (considered as constant).

This paper focuses on the average hop count measure $\mathcal{H}_{\text{avg}}$ of a digraph, and explores the variant of the VPL problem which seeks to optimize this measure. (Actually, for convenience, we formulate our results in terms of the equivalent *total hop count* measure, $\mathcal{H}_{\text{tot}}(H)$.) Clearly, the *upper* bounds established in [3] can be converted into ones for the *average* hop count $\mathcal{H}_{\text{avg}}(H)$ of DVPL's as well. However, the average hop count may admit better bounds.

While most studies of the VPL problem considered all-to-all instances, another pattern of communication demands for which the VPL problem was studied is the *one-to-all* pattern, in which a single vertex must communicate with all other vertices [7,9]. For the chain network, a duality between the problem of minimizing the hop count given a bound c on the maximum load and the problem of minimizing the load given a bound h on the maximum hop count, is established in [7]. A number of VPL optimization problems with one-to-all communication demands are studied in [9] for the undirected chain network, including the problem of designing a VPL with optimal average hop count given a maximum load bound c under the one-to-all communication pattern, and a dynamic programming algorithm is presented for this problem. The "Open problems" section of [9] states the following:

> "The most immediate open problem is to generalize these results for arbitrary trees, a task which seems nontrivial, as far as the dynamic programming algorithms are concerned, due to the additional structural information that is attached to each subtree (which does not exist in chains)."

The main result of the current paper, presented in Section 3, makes progress towards solving the open problem mentioned in the above quote. On the one hand, our solution applies only to directed trees, with all the edges directed away from the root. On the other hand, our solution is slightly more general, since instead of assuming a uniform constant capacity c on all links, we allow a capacity function $c : E \mapsto \mathbb{N}^+$, under the restriction that $c(e) \leqslant c_0$ for every link e, for some constant c_0. Moreover, we deal with a general class of one-to-all instances, referred to as *one-to-many* instances, in which the root must communicate with some partial subset of the vertices. Actually, as shown in Section 4, our algorithm can be modified to handle the *weighted* version of the problem [9] in which given a *requirements matrix* $\widehat{R}$, such that for $1 \leqslant i \leqslant n$, $\widehat{R}_{1i}$ specifies the expected amount of traffic between v_1 and v_i, it is necessary to minimize the weighted average hop count of G, defined by weighing the hop counts according to $\widehat{R}$.

2. Preliminaries

A communication network is represented by an n-vertex connected weighted directed graph (digraph) $G = (V, E)$ with a weight function c on the edges. The vertex set V represents the network switches, and the arc set $E \subseteq V \times V$ represents the set of physical directed links. The weight function assigns each arc e its capacity $c(e)$. The arc $\langle v, u \rangle$ is said to be an *incoming* arc of u.

For two vertices $v, u \in V$, the *hop count* (or *distance*) from v to u in G, denoted $d_G(v, u)$ (or just $d(v, u)$ when the context is clear), is the length of the shortest path from v to u in G. Note that in a directed graph it may happen that $d_G(v, u) \neq d_G(u, v)$.

Given a communication network, it is possible to assign to certain vertex pairs $x, y \in V$ a simple directed path (dipath) $P(x, y)$ connecting x to y. Let $E' \subseteq V \times V$ be the set of the vertex pairs (x, y) for which such a dipath $P(x, y)$ is defined. The digraph $H = (V, E')$ is called a *virtual digraph* of G. The path $P(x, y)$ in the original graph G associated with the arc $\langle x, y \rangle$ in H is called a *virtual path* (*VP*). The pair (H, P) is a *directed virtual path layout* (*DVPL*) of G and an arc of H is a *virtual arc*. With each dipath $Q = (e'_1, \ldots, e'_l)$ in H we associate a *route* in G consisting of the concatenation of $P(e'_1), \ldots, P(e'_l)$.

The *load* of an arc e of G is the number of virtual arcs e' of H whose associated virtual dipaths $P(e')$ contain the arc e, that is, $l(e) = |\{e' \in E' \mid e \in P(e')\}|$. A DVPL (H, P) satisfying the requirement $l(e) \leqslant c(e)$ for every $e \in E$ is referred to as a *c-admissible directed virtual path layout* (*c-DVPL*) of G. For a digraph G with capacity function c, denote by $\mathrm{Virt}(G, c)$ the set of all c-admissible DVPLs of G. The aim is to design a c-DVPL of G which satisfies some constraints on the length of the virtual dipaths. This corresponds to constraining some measures of the DVPL.

In some situations, it is known in advance that certain pairs of nodes are not expected to communicate with each other. We model such situations by associating with the graph a set of *communication requirements*, i.e., a set of pairs of nodes which need to communicate. Formally, for an n-vertex (directed or undirected) graph $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$, the *requirements matrix* R is an $n \times n$ boolean matrix such that for $1 \leqslant i, j \leqslant n$, $R_{ij} = 1$ if v_i and v_j need to communicate with each other, and 0 otherwise. We assume that each element in the diagonal of a requirements matrix is zero.

Let $G = (V, E)$ be a directed n-vertex graph, and let R be a requirements matrix. The *R-total hop count* is defined as

$$\mathcal{H}_{\mathrm{tot}}(G, R) = \sum_{R_{ij}=1} d_G(v_i, v_j).$$

Let us now turn to dealing with virtual digraphs and DVPLs. Consider an n-vertex graph G with a requirements matrix R and a capacity function c. Define the *minimal realizable total hop count* of (G, c) under R, denoted $\widetilde{\mathcal{H}}_{\mathrm{tot}}(G, c, R)$, as the minimum R-total hop count that can be achieved by such a virtual digraph, i.e.,

$$\widetilde{\mathcal{H}}_{\mathrm{tot}}(G, c, R) = \min\bigl\{\mathcal{H}_{\mathrm{tot}}(H, R) \mid (H, P) \in \mathrm{Virt}(G, c)\bigr\}.$$

In what follows we consider the algorithmic problem of finding a c-admissible DVPL which minimizes the R-total hop count, as well as calculating the minimal realizable R-total hop count, for a given pair (G, c). This problem may be complicated in general, but as shown in this paper it can be solved for trees, under some natural constraints on the allowed communication requirements matrix R.

Specifically, we take G to be an n-vertex rooted tree, with root v_1 and all arcs directed away from the root. Suppose that all the required communication is for the root to communicate with some other vertices, i.e., the entries in R are all zero, except for the first row, which has a 1 in some entries. The vertices with which the root needs to communicate are called the *required destinations*, and are denoted by $Q(R) = \{v_i \mid R_{1i} = 1\}$. We also

assume that the capacity function c is *bounded* by some constant $c_0 \in \mathbb{N}$. The goal is to calculate $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, R)$, and find a c-admissible DVPL of G that achieves it.

Note. Over tree networks, the virtual path associated with each virtual edge is uniquely defined by the virtual digraph H and the tree, hence a DVPL (H, P) can be fully specified by specifying the virtual digraph H. Thus for notational simplicity we hereafter ignore the component P of a DVPL and refer only to H.

3. An algorithm for one-to-many requirements on trees

3.1. The dynamic programming paradigm

For a given tree G and a set $Q(R)$ of required destinations, we show that the best DVPL can be found by dynamic programming. For this, we would like to evaluate the contribution of a subtree to the total hop count. Let the vertices of G other than the root be $v_2, \ldots, v_n$, organized in breadth-first order, i.e., satisfying that if v_i is the parent of v_j, then $i < j$. Each v_i defines a subtree of G, denoted T_i, whose set of vertices V_i includes v_i and all the vertices below it in the tree. Also denote by e_i the arc entering v_i. For some c-DVPL H of G, and some subtree T_i of G, define the *internal total hop count of T_i with respect to H* as

$$\text{InTot}(H, T_i, R) = \sum_{v_j \in V_i \cap Q(R)} d_H(v_1, v_j).$$

Also define the *minimal realizable total hop count* of T_i over all such c-DVPLs as

$$\widetilde{\mathcal{H}}_{\text{tot}}(T_i, R) = \min\bigl\{\text{InTot}(H, T_i, R) \mid (H, P) \in \text{Virt}(G, c)\bigr\}.$$

As usual, the dynamic programming algorithm is based on solving many small subproblems gradually. The subproblem $\Psi(T_i, \vec{d})$ to be solved by the algorithm is defined as follows:

Subproblem $\Psi(T_i, \vec{d})$

Input: (1) A subtree T_i rooted at v_i.

(2) A tuple $\vec{d} = \langle d_1, \ldots, d_k \rangle$ of k nonnegative integers.

Intuitively, the tuple represents the starting points of $k \geqslant 0$ virtual arcs $\mathcal{A} = \langle A_1, \ldots, A_k \rangle$ entering T_i. Each arc A_j starts at some node x_j on the path between v_1 and v_i. The input component d_i represents the hop count in H of its start vertex x_j from v_1, namely $d_j = d_H(v_1, x_j)$. This value is henceforth referred to as the *tail-length* of A_j. The end vertex of each of the k arcs is some (unknown) vertex y_j of T_i (see Fig. 1). Hence these virtual arcs are not yet completely specified, as their end-points y_j are to be chosen by the solution devised for the subproblem Ψ. It is convenient to refer to the virtual arcs of $\mathcal{A}$ throughout the following discussion, but the reader should bear in mind that these arcs are only implicit in the algorithm, and only the $\vec{d}$ vectors are manipulated explicitly.

Output: A value $f(T_i, \vec{d})$, which is a nonnegative integer or ∞.

For a subtree T_i rooted at v_i and a tuple $Y = \langle y_1, \ldots, y_k \rangle$ of end vertices of the virtual arcs, denote by H_Y the virtual digraph of T_i including all *internal* virtual arcs (i.e., arcs of H downwards from vertices of T_i), and the k paths from v_1 to each y_j of lengths

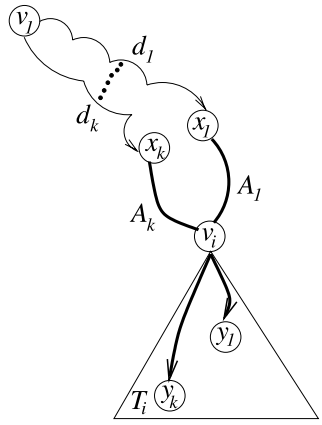

Fig. 1. A typical problem to be solved.

$d_j + 1$, for $1 \leqslant j \leqslant k$. Let Y^* denote the optimal tuple Y, such that the internal total hop count $\mathrm{InTot}(H_{Y^*}, T_i, R)$ of T_i is minimal over all c-DVPL including k paths from v_1 to some vertices in T_i of lengths $d_j + 1$. Denote this minimal internal total hop count for given T_i, R and $\vec{d}$ by $\mathrm{OptInTot}(T_i, R, \vec{d})$. The output of $\Psi(T_i, \vec{d})$, denoted by $f(T_i, \vec{d})$, is $\mathrm{OptInTot}(T_i, R, \vec{d})$ if this can be done, or ∞ if the instance $(T_i, \vec{d})$ is infeasible.

We calculate $\widetilde{\mathcal{H}}_{\mathrm{tot}}(G, c, R)$ (and find the c-DVPL that achieves it), by solving all the subproblems $\Psi(T_i, \vec{d})$, namely, finding $f(T_i, \vec{d})$ for every T_i and every possible tuple of tail-lengths $\vec{d}$. This is done according to the dynamic programming paradigm. We create a table F of n rows and $\frac{n^{c_0+1}-1}{n-1}$ columns. The columns are grouped into sets $F_0, F_1, \ldots, F_{c_0}$, where F_k consists of n^k columns. These columns are used to store all possible outputs $f(T_i, \vec{d})$. Row i of the table stands for the subtree T_i rooted at the vertex v_i. Note that the rows are ordered $0, 1, \ldots, n-1$, so if v_i is an ancestor of v_j, then T_i's row is above T_j's row. The entries of this set of columns and of the ith row stand for the class of subproblems in which k virtual arcs enter T_i.

As $d_H(v_1, x_j) \leqslant n-1$, for every $x_j \in V$ and every virtual digraph H including a path from v_1 to x_j, the kth set has n^k columns, each standing for one tuple of $\vec{d}$. The procedure will store $f(T_i, \vec{d})$ in the entry of the ith row and the column corresponding to $\vec{d}$.

Clearly, if $k = 0$, and T_i includes any required destinations, then the subproblem $\Psi(T_i, \vec{d})$ for $\vec{d} = \langle\,\rangle$ is infeasible, so the value for this entry is ∞. Note that if the (physical) arc reaching v_i is e_i, then the subproblem $\Psi(T_i, \vec{d})$ for $|\vec{d}| > c(e_i)$ is also infeasible, so the columns in sets F_k for $k > c(e_i)$ (if there are any) are irrelevant, and should also be filled with ∞. Note also that for each T_i there is no need to use more virtual arcs than the number of required destinations in T_i, but we do take such cases into consideration during the table filling.

Note that if $k \geqslant 1$ arcs reach v_i, then it is never necessary to terminate more than one of them at v_i itself. Hence we can extend some (perhaps all) of them to reach some internal vertices of T_i. The way we do it affects the needed internal total hop count.

3.2. The algorithm

The algorithm proceeds as follows. The table is filled in a dynamic programming manner, from the bottom up, filling each row by using the already filled ones below it.

For a leaf vertex v_i (for which the subtree T_i consists of v_i alone), filling the relevant entries in the table is done as follows. If v_i is not a required destination, then $f(T_i, \langle d_1, \ldots, d_k\rangle) = 0$. If it is, then there are two cases: if $0 < k \leqslant c(e_i)$, then $f(T_i, \langle d_1, \ldots, d_k\rangle) = \min_{1\leqslant j\leqslant k}\{d_j\} + 1$; else, the subproblem $\Psi(T_i, \vec{d})$ is infeasible, and $f(T_i, \langle d_1, \ldots, d_k\rangle) = \infty$. In summary,

$$f(T_i, \vec{d}) = \begin{cases} 0, & v_i \notin Q(R), \\ \min_{1\leqslant j\leqslant k}\{d_j\} + 1, & v_i \in Q(R) \text{ and } 0 < k \leqslant c(e_i), \\ \infty, & \text{otherwise.} \end{cases}$$

Suppose we now have to fill row i for a nonleaf v_i, and suppose T_i has m subtrees, $T_{i_1}, \ldots, T_{i_m}$, where rows $i_1, \ldots, i_m > i$ of the table are already filled. Consider a subproblem $\Psi(T_i, \vec{d})$ for $\vec{d} = \langle d_1, \ldots, d_k\rangle$. We need to solve the subproblem $\Psi(T_i, \vec{d})$ of row i, having the subproblems $\Psi(T_{i_1}, \vec{d}^1), \ldots, \Psi(T_{i_m}, \vec{d}^m)$ already solved for all relevant possible values of $\vec{d}^1, \ldots, \vec{d}^m$. For this, we need to check all the possibilities of partitioning the arc tuple $\mathcal{A}$ into subtuples of arcs which reach the subtrees T_{i_j}, considering how the appropriate subproblems are solved accordingly.

Again, if $|\vec{d}| = k = 0$ and yet T_i includes any required destination, then the problem is infeasible and we have $f(T_i, \vec{d}) = \infty$. On the other hand, if $k = 0$ and T_i includes no required destinations, then return $f(T_i, \vec{d}) = 0$.

Hereafter, assume that $|\vec{d}| = k > 0$. The value for $\Psi(T_i, \vec{d})$ depends on whether v_i is a required destination or not. Let us first consider the case where v_i is a required destination. In this case, we need one virtual arc to end at v_i. So we have to partition the tuple $\vec{d}$ of k incoming arc tail-lengths into $m + 1$ disjoint arc tuples $\vec{d}^0, \vec{d}^1, \ldots, \vec{d}^m$ of (nonnegative) cardinalities $1, k_1, \ldots, k_m$ (some of which may be zero) so that $\sum_{j=1}^{m} k_j = k - 1$.

We refer to such a partition of $\vec{d}$ as a k^+-*partition*, and denote it by $X = \langle d_0, \vec{d}^1, \ldots, \vec{d}^m\rangle$. For every $1 \leqslant j \leqslant m$, we denote the number of elements of the component $\vec{d}^j$ of X by $k_j(X)$, and denote this component by $\vec{d}^j(X) = \langle d_1^j(X), \ldots, d_{k_j(X)}^j(X)\rangle$. The set of all k^+-partitions will be denoted as $\mathcal{P}^+(T_i, \vec{d})$. Every such k^+-partition X satisfies $\bigcup_{j=1}^{m} \vec{d}^j(X) \cup \{d_0(X)\} = \vec{d}$. (For notational simplicity we refer here to each tuple as a multiset of its elements. Note that it may happen that $\vec{d}^j(X)$ is empty, so $k_j(X)$ is zero.)

Each such partition $X = \langle d_0(X), \vec{d}^1(X), \ldots, \vec{d}^m(X)\rangle$ corresponds to one possibility of extending the arcs reaching v_i to some of its descendants. The $\mathcal{A}_j(X)$ corresponding to $\vec{d}^j(X)$ is the set of arcs to be extended to T_{i_j}, and $\vec{d}^j(X)$ is the tuple of tail-lengths, namely, hop counts from v_1 to the vertices from which the virtual arcs of $\mathcal{A}_j(X)$ continue into T_{i_j}. (Again, let us stress that the algorithm maintains and manipulates explicitly only the vectors $\vec{d}^j$, and the tuples $\mathcal{A}_j$ are only implicit.)

For every $1 \leqslant j \leqslant m$, we can also add $t_j(X) \leqslant c(e_{i_j}) - k_j(X)$ *new* virtual arcs, starting at v_i and going to v_{i_j}. We denote the resulting tuple of hop counts of virtual arcs entering

T_{i_j} by

$$\vec{d}^{\,j}\big(t_j(X), X\big) = \big\langle d_1^j(X), \ldots, d_{k_j(X)}^j(X), \underbrace{d_0(X)+1, \ldots, d_0(X)+1}_{t_j(X) \text{ times}}\big\rangle.$$

For each $1 \leqslant j \leqslant m$, we use the *already solved* subproblems $\Psi(T_{i_j}, \vec{d}^{\,j}(t_j(X)), X)$, for every $0 \leqslant t_j(X) \leqslant c(e_{i_j}) - k_j(X)$, and take $k'_j(X)$ to be the $t_j(X)$ value that achieves the minimal total hop count for T_{i_j} under the partition X, namely, such that

$$f\big(T_{i_j}, \vec{d}^{\,j}(k'_j(X), X)\big) = \min_{0 \leqslant t_j(X) \leqslant c(e_{i_j}) - k_j(X)} \big\{ f\big(T_{i_j}, \vec{d}^{\,j}\big(t_j(X), X\big)\big)\big\}.$$

Note that fixing $T_i, \vec{d}$ and the partition X determines the optimal numbers $k'_1(X), \ldots, k'_m(X)$ of new virtual arcs to be added from v_i into $T_{i_1}, \ldots, T_{i_m}$ respectively. Hence the actual number of virtual arcs entering each T_{i_j} under X is $\hat{k}_j(X) = k_j(X) + k'_j(X)$. Hereafter, we denote for $1 \leqslant j \leqslant m$,

$$\vec{d}^{\,j}(\widehat{X}) = \vec{d}^{\,j}\big(\hat{k}_j(X), X\big).$$

Hence assuming the particular k^+-partition X, and letting H^* be the best c-DVPL with respect to $T_i, \vec{d}$, and X, the total hop count of H^*, denoted $g(T_i, \vec{d}, X)$, consists of the hop count from v_1 to the node v_i itself, which is $d_0 + 1$, plus the sum of the optimal total hop counts to all the required destinations in each of the m subtrees T_{i_j} under X. Hence

$$g(T_i, \vec{d}, X) = d_0 + 1 + \sum_{j=1}^{m} f\big(T_{i_j}, \vec{d}^{\,j}(\widehat{X})\big).$$

Now $f(T_i, \vec{d})$ is taken to be the minimum over all k^+-partitions X of $g(T_i, \vec{d}, X)$,

$$f(T_i, \vec{d}) = \min_{X \in \mathcal{P}^+(T_i, \vec{d})} \big\{ g(T_i, \vec{d}, X)\big\}.$$

Let us now turn to the case where v_i is not a required destination. In this case $f(T_i, \vec{d})$ is calculated similarly, with some minor changes. In principle, we have to examine two possibilities, namely, the possibility that one of the virtual arc terminates at v_i, and the possibility that none does. To cover the first possibility, we simply follow the same process described above for the previous case, except that we define

$$g_1(T_i, \vec{d}, X) = \sum_{j=1}^{m} f\big(T_{i_j}, \vec{d}^{\,j}(\widehat{X})\big)$$

(as the length of the path reaching v_i is not taken into account), and

$$f_1(T_i, \vec{d}) = \min_{X \in \mathcal{P}^+(T_i, \vec{d})} \big\{ g_1(T_i, \vec{d}, X)\big\}.$$

This produces a candidate value, $f_1(T_i, \vec{d})$. Next, we look for the best solution among those following the second possibility, namely, those partitions in which no virtual arcs terminates at v_i. In this case we have to examine all possible partitions of the tuple $\vec{d}$

of k incoming arcs into m tuples of tail-lengths $\vec{d}^1, \ldots, \vec{d}^m$ of (nonnegative) cardinalities $k_1, \ldots, k_m$ (some of which may be zero), so that $\sum_{j=1}^{m} k_j = k$. We call this a *k-partition* denoted by $X = \langle \vec{d}^1, \ldots, \vec{d}^m \rangle$. As in the previous case, for every $1 \leqslant j \leqslant m$, we denote the number of elements of the jth component of X by $k_j(X)$, denote this component by $\vec{d}^j(X) = \langle d_1^j(X), \ldots, d_{k_j(X)}^j(X) \rangle$, and also denote the set of all k-partitions as $\mathcal{P}(T_i, \vec{d})$. Such a partition must satisfy $\bigcup_{j=1}^{m} \vec{d}^j(X) = \vec{d}$.

Such a partition corresponds to a possibility of extending the virtual arcs entering v_i to some of its descendants, while having no virtual arc terminating at v_i.

Now we define $g_2(T_i, \vec{d}, X) = \sum_{j=1}^{m} f(T_{i_j}, \vec{d}^j(X))$, and $f_2(T_i, \vec{d})$ is taken to be the minimum, over all k-partitions X, of $g_2(T_i, \vec{d}, X)$, i.e., $f_2(T_i, \vec{d}) = \min_{X \in \mathcal{P}(T_i, \vec{d})} \{g_2(T_i, \vec{d}, X)\}$. Finally, we take $f(T_i, \vec{d})$ to be the best of these two cases, $f(T_i, \vec{d}) = \min\{f_1(T_i, \vec{d}), f_2(T_i, \vec{d})\}$.

For finding the minimal realizable total hop count of the tree G, suppose the root v_1 has s subtrees, $T_{1_1}, \ldots, T_{1_s}$. For every $0 \leqslant k \leqslant c_0$, define

$$\vec{d}^{(k)} = \langle \underbrace{0, \ldots, 0}_{k \text{ times}} \rangle,$$

and for every $1 \leqslant j \leqslant s$, define

$$\Upsilon(T_{1_j}) = \min_{0 \leqslant k \leqslant c_0} \left\{ f(T_{1_j}, \vec{d}^{(k)}) \right\},$$

and the minimal realizable total hop count is taken to be $\sum_{j=1}^{s} \Upsilon(T_{1_j})$.

3.3. Analysis

Now we prove that the algorithm indeed calculates $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, R)$, and bound its complexity.

For proving correctness, we first observe that the minimal realizable total hop count of G, $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, R)$, is the minimal internal total hop count of v_1's subtree, i.e., $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, R) = \widetilde{\mathcal{H}}_{\text{tot}}(T_1, R)$. Next we observe that the minimum for each subtree is independent on the other subtrees. As the algorithm takes $\widetilde{\mathcal{H}}_{\text{tot}}(T_1, R)$ as the sum of $\Upsilon(T_{1_1}), \ldots, \Upsilon(T_{1_t})$, we need to prove that each $\Upsilon(T_{1_j})$ achieves the relevant minimal internal total hop count. Given the definition of $\Upsilon(T_{1_j})$, we should prove that $f(T_i, \vec{d})$ indeed gives the minimal internal total hop count of T_i, being reached by k virtual arcs with a tuple $\vec{d}$ of tail-lengths from v_1.

Lemma 3.1. *For every $1 \leqslant i \leqslant n$ and tuple $\vec{d} = \langle d_1, \ldots, d_k \rangle$, $f(T_i, \vec{d}) = \text{OptInTot}(T_i, R, \vec{d})$.*

Proof. We prove this by induction on T_i's depth. For a leaf subtree, by the rule for filling the table entries in leaf rows, the result is $f(T_i, \vec{d}) = \min_{1 \leqslant j \leqslant k} \{d_k\}$ for a required destination, and 0 for a non-required destination, which is definitely true. (Note that this gives $\Upsilon(T_i) = 1$ for a leaf subtree which is a required destination, and $\Upsilon(T_i) = 0$ for a leaf subtree which is not.)

Now suppose $f(T_i, \vec{d})$ gives the required minimum for a subtree T_i of depth no greater than d. Let T_i be a subtree of depth $d + 1$, rooted at v_i, and let $v_{i_1}, \ldots, v_{i_m}$ be v_i's direct

descendants, and their subtrees be $T_{i_1}, \ldots, T_{i_m}$ respectively. Consider the entry corresponding to the instance $(T_i, \vec{d})$ in the table. Suppose there are k virtual arcs reaching T_i. If v_i is not a required destination, then there are two subcases. First, it may be the case that none of the k arcs stop at v_i, and they all go to $T_{i_1}, \ldots, T_{i_m}$. Then we need to find the best way to distribute the arcs, according to the results of f for $T_{i_1}, \ldots, T_{i_m}$. The second subcase is that some of the arcs stop at v_i, where the rest are destined for nodes in $T_{i_1}, \ldots, T_{i_m}$, and other arcs can go from v_i to some v_{i_j}. Now we are required to find which arc stops at v_i, how the other arcs are destined for nodes in $T_{i_1}, \ldots, T_{i_m}$, and whether there are other arcs going from v_i to some v_{i_j}. All this is also decided according to the results of f for $T_{i_1}, \ldots, T_{i_m}$.

Now, in each case we choose the combination that achieves the minimal sum of internal total hop counts, going over *all* combinations possible in the situation, according to the values of f for $T_{i_1}, \ldots, T_{i_m}$ and some appropriate tail-length tuples $\vec{d}^j$ for $1 \leqslant j \leqslant m$. The induction hypothesis states that these values are indeed the minimal internal total hop count values, i.e., that $f(T_{i_j}, \vec{d}^j) = \text{OptInTot}(T_{i_j}, R, \vec{d}^j)$ for every $1 \leqslant j \leqslant m$, so in each case we get the needed result. Choosing the minimal of all possibilities, we get the minimal internal total hop count value for T_i reached by those k arcs.

The case in which v_i is a required destination is quite the same, except that the collection of cases that need to be examined is slightly more limited, since there must be at least one arc terminating at v_i, and the hop count of v_i should be taken into account. Therefore, this case is treated precisely as a version of the second subcase of the previous case. □

As for the complexity of the algorithm, note that there are $(n-1) \cdot \frac{n^{c_0+1}-1}{n-1} = \Theta(n^{c_0+1})$ entries in the table. For each entry, related to some possible pair $(T_i, \vec{d})$, for $k \leqslant c_0$, we need to check many possible cases, corresponding to the particular possible ways of partitioning a tuple $\vec{d}$ of k elements into up to n sets (some of which may be empty), directed at v_i's children, and completing each set to a total of up to c_0 virtual arcs by adding new virtual arcs (originating at v_i). The number of possibilities is, therefore, $O(n^{c_0})$. Over all, the algorithm's complexity is $O(n^{2c_0+1})$, which is polynomial in n for any fixed bound c_0.

The algorithmic procedure described above calculates $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, R)$. For finding the c-DVPL which achieves it, we need to modify the algorithm a little, as follows. Each time we solve $\Psi(T_i, \vec{d})$, we add to its entry the *partition tuple* X describing the distribution of the k arc tail lengths among $T_{i_1}, \ldots, T_{i_m}$ achieving the best internal total hop count $f(T_i, \vec{d})$. The tuple is of length k, indicating, for every arc reaching v_i, the subtree of T_i it is extended to, or a stop at v_i. As described above, X is actually computed during the calculation of $f(T_i, \vec{d})$, in accordance with the chosen k or k^+-partitions, so no additional work is required. After $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, R)$ is computed, we first find the number of virtual arcs going from v_1 to each of its subtrees. Then we discover the desired c-DVPL by keeping track over the tree's vertices in a breadth-first manner. Reaching vertex v_i with k arcs, having tail-lengths $d_1, \ldots, d_k$, we look at the partition tuple X of $\Psi(T_i, \langle d_1, \ldots, d_k \rangle)$ in our table, and find the arcs that stop at v_i (if any), and the distribution of the others among its direct descendants. Along our tour, we memorize each vertex we come across as the start, the end or the passing through vertex of the relevant virtual arcs, so after visiting every vertex, we have the desired virtual digraph H.

Therefore, we proved the following theorem.

Theorem 3.2. *The algorithm described above calculates* $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, R)$*, and finds the* c*-DVPL that achieves it in polynomial time.*

4. Handling generalized one-to-many requirements

In the previous section we dealt with requirements in the form of a boolean matrix R identifying the set of vertex pairs in the graph which need to communicate. We now generalize this notion by addressing the possibility that the communication between a pair of nodes may be of some priority. We do not categorize each pair of nodes as "need to communicate" or "need not", but rather assign to it a nonnegative number between 0 and 1, which indicates the *expected amount of traffic* between those nodes.

For an n-vertex (directed or undirected) graph $G = (V, E)$ whose vertices are $v_1, \ldots, v_n$, define the *general requirements matrix*, $\widehat{R}$, as the $n \times n$ matrix such that for $1 \leqslant i, j \leqslant n$, $\widehat{R}_{ij}$ is the expected amount of traffic from v_i to v_j. As before, we assume that each element in the diagonal of a general requirements matrix is zero. For the pair $(G, \widehat{R})$, define the $\widehat{R}$*-total hop count* of G with respect to the requirements as

$$\mathcal{H}_{\text{tot}}(G, \widehat{R}) = \sum_{1 \leqslant i, j \leqslant n} d_G(v_i, v_j) \cdot \widehat{R}_{ij}.$$

Next, we turn to c-DVPLs of graphs. For an n-vertex graph G with a general requirements matrix $\widehat{R}$, and a capacity function c, define $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, \widehat{R})$ as the minimum $\widehat{R}$-total hop count that can be achieved by a c-DVPL of G.

Now we generalize our algorithm for 0–1 one-to-many requirement matrices to general one-to-many requirement matrices (i.e., matrices where all the entries are zero except for some entries in the first row, which may be arbitrary nonnegative integers). We show that the problem of calculating the total hop count can be solved for trees, under some constraints on the allowed (general) one-to-many requirements matrix $\widehat{R}$.

The notation and the algorithm are a natural generalization of those of the previous section, with minor changes. Having a general requirements matrix $\widehat{R}$ for the tree graph G, for some c-DVPL H of G, and some subtree T_i the *internal total hop count of* T_i *with respect to* H is defined with respect to $\widehat{R}$:

$$\text{InTot}(H, T_i, \widehat{R}) = \sum_{v_j \in V_i} d_H(v_1, v_j) \cdot \widehat{R}_{1j},$$

and the *minimal realizable total hop count of* T_i is defined accordingly:

$$\widetilde{\mathcal{H}}_{\text{tot}}(T_i, \widehat{R}) = \min\bigl\{\text{InTot}(H, T_i, \widehat{R}) \mid (H, P) \in \text{Virt}(G)\bigr\}.$$

The dynamic program calculations and the table filling are done as in the previous section, only filling leaf vertices' values with respect to $\widehat{R}$:

$$f(T_i, \vec{d}) = \begin{cases} 0, & k = 0 \text{ and } \widehat{R}_{1i} = 0, \\ \infty, & k = 0 \text{ and } \widehat{R}_{1i} \neq 0, \\ \min_{1 \leqslant j \leqslant k}\{(d_j + 1) \cdot \widehat{R}_{1i}\}, & 0 < k \leqslant c(e_i), \\ \infty, & k > c(e_i). \end{cases}$$

Also taking the communication requirements into consideration when calculating the function g:

$$g(T_i, \vec{d}, X) = (d_0 + 1) \cdot \widehat{R}_{1i} + \sum_{j=1}^{m} f\left(T_{i_j}, \vec{d}^{\,j}(\widehat{X})\right).$$

The correctness proof and complexity analysis for the algorithm remain the same, and so does the modification which enables us to find the optimal c-DVPL. We thus have the following.

Theorem 4.1. *The algorithm described above calculates $\widetilde{\mathcal{H}}_{\text{tot}}(G, c, \widehat{R})$, and finds the c-DVPL that achieves it in polynomial time.*

References

[1] S. Ahn, R.P. Tsang, S.R. Tong, D.H.C. Du, Virtual path layout design on ATM networks, in: Proc. IEEE INFOCOM'94, 1994, pp. 192–200.
[2] L. Becchetti, P. Bertolazzi, C. Gaibisso, G. Gambosi, On the design of efficient ATM routing schemes, Theoret. Comput. Sci. 270 (2002) 341–359.
[3] J.C. Bermond, N. Marlin, D. Peleg, S. Perennes, Directed virtual path layouts in ATM networks, Theoret. Comput. Sci. 291 (2003) 3–28.
[4] P. Chanas, O. Goldschmidt, Conception de réseau de VP de diamètre minimum pour les réseaux ATM, in: Road-f'98, 1998, pp. 38–40.
[5] I. Cidon, O. Gerstel, S. Zaks, A scalable approach to routing in ATM networks, in: Proc. 8th Int. Workshop on Distributed Algorithms (WDAG), in: Lecture Notes in Comput. Sci., vol. 857, Springer-Verlag, Terschelling, The Netherlands, 1994, pp. 209–222.
[6] T. Eilam, M. Flammini, S. Zaks, A complete characterization of the path layout construction problem for ATM networks with given hop count and load, Parallel Process. Lett. 8 (1998) 207–220.
[7] M. Feighelstein, S. Zaks, Duality in chain ATM virtual path layouts, in: Proc. 4th Int. Colloq. on Structural Information and Communication Complexity (SIROCCO), Monte Verita, Ascona, Switzerland, 1997, pp. 228–239.
[8] O. Gerstel, I. Cidon, S. Zaks, The layout of virtual path in ATM networks, IEEE Trans. Networking 4 (6) (1996) 873–884.
[9] O. Gerstel, A. Wool, S. Zaks, Optimal layouts on a chain ATM network, Discrete Appl. Math. 83 (1998) 157–178.
[10] O. Gerstel, S. Zaks, The virtual path layout problem in fast networks, in: Proc. 13th ACM Symp. on Principles of Distributed Computing (PODC), Los Angeles, CA, 1994, pp. 235–243.
[11] R. Händler, M.N. Huber, Integrated Broadland Networks: An Introduction to ATM-based Networks, Addison-Wesley, Reading, MA, 1991.
[12] ITU recommendation, I series (B-ISDN), Blue Book, 1990.
[13] E. Kranakis, D. Krizanc, A. Pelc, Hop-congestion tradeoffs for high-speed networks, Internat. J. Found. Comput. Sci. 8 (1997) 117–126.
[14] F.Y.S. Lin, K.T. Cheng, Virtual path assignment and virtual circuit routing in ATM networks, in: Proc. IEEE GLOBECOM'93, 1993, pp. 436–441.
[15] C. Partridge, Gigabit Networking, Addison-Wesley, Reading, MA, 1994.
[16] S. Zaks, Path Layout in ATM networks—a survey, in: The DIMACS Workshop on Networks in Distributed Computing, DIMACS Center, Rutgers University, 1997.